JAVA SE

Quiz yourself: Rules about throwing checked exceptions in Java

What happens when you throw a checked exception? There are some tricky special cases.

by Mikalai Zaikin and Simon Roberts

April 19, 2021

This quiz focuses on throwing checked exceptions. Given the following class

```
public class ArgsChecker {
  public static void main(String[] args) {
    try {
      if (args.length == 0) {
        throw new IllegalArgumentException();
      }
    } catch (NullPointerException e) {
      System.out.print("NPE ");
      throw e;
    } catch (IllegalArgumentException e) {
      System.out.print("IAE ");
      throw e;
    } catch (Throwable t) {
      System.out.print("T ");
      throw t; // line n1
    } finally {
      System.out.print("F ");
    }
  }
}
```

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

Which of the following is correct if you compile and run the `ArgsChecker` class with no arguments? Choose one.

A. The class compiles, runs, and prints `NPE T F` followed by a stack trace.

The answer is A.

B. The class compiles, runs, and prints `IAE T F` followed by a stack trace.

The answer is B.

C. The class compiles, runs, and prints `IAE F` followed by a stack trace.

The answer is C.

D. The class does not compile. If you remove line n1, it compiles, runs, and prints `NPE T F` followed by a stack trace.

The answer is D.

E. The class does not compile. If you remove line n1, it compiles, runs, and prints `IAE T F` followed by a stack trace.

The answer is E.

F. The class does not compile. If you remove line n1, it compiles, runs, and prints `IAE F` followed by a stack trace.

The answer is F.

Answer. When you look at the code, you should notice that the `main` method's declaration does not include the word `throws`. This means that `main` must either handle all the checked exceptions that might arise in the body of the method, or it will fail to compile. Notice that this comment is specific to *checked* exceptions, because any method is permitted to throw any *unchecked* exception without any requirement for a `throws` declaration announcing the possibility.

In view of the fact that half the quiz options suggest the code might not compile, it makes sense to investigate this possibility first.

Both `NullPointerException` and `IllegalArgumentException` are subclasses of `RuntimeException` and, as such, are unchecked. The only exception that's thrown in the `try` block is `IllegalArgumentException`. You can safely ignore the code that rethrows these exceptions for this part of the investigation.

The code on line n1 says `throw t`, and the variable `t` is of type `Throwable`. `Throwable` is a checked exception, but the `main` method does not declare `throws` as `Throwable`. It wouldn't be unreasonable to think this means the code will not compile. In fact, in Java 6 or previous versions, this code would have been rejected for exactly this reason. However, since Java 7, the code is perfectly correct.

Why? The explanation (as you would expect) can be found in the Java specification, such as the specification for Java SE 11. The relevant part of section 11.2.2 of the specification, "Exception analysis of statements," says

> A `throw` statement whose thrown expression is a final or effectively final exception parameter of a `catch` clause `C` can throw an exception class `E` iff:
>
> `E` is an exception class that the `try` block of the `try` statement which declares `C` can `throw`…

The wording is perhaps more factual than explanatory, so let's explore it here. Under specific conditions, the exceptions that must be declared by a method are those that can actually arise in the `try` block, rather than those of the type of the rethrown exception. Notice that in this question, there are no checked exceptions that might arise in the `try`. (Remember that iff means "if and only if.")

The condition under which this rule applies is that the formal parameter to the `catch` block, `Throwable t` in this case, must be either final or effectively final, and it must be the expression that is rethrown. In the question, there is no assignment to `t`, so it is effectively final, and the

`throw` statement is simply `throw t`. This all means that the special case conditions are met here, and there is no need for the `main` method to declare any checked exceptions. Based on this you can say that line n1 compiles successfully. Therefore, options D, E, and F are incorrect.

To better understand this special case, here are a couple of experiments. First, change the code to the following:

```
try {
    if (args.length == 0) {
        throw new Exception();
    }
}
```

In this case, the checked exception (`Exception`) might arise in the `try` block and the compiler will reject the code unless you also modify the `main` method so that it declares `throws Exception`.

A separate experiment is to change the code, as follows:

```
} catch (Throwable t) {
    System.out.print("T ");
    Throwable t1 = t;
    t = t1;
    throw t; // line n1
} finally {
```

This time, you're throwing the exact same exception, with the exact same expression, but the variable is no longer effectively final. Therefore, the special case does not apply. Consequently, the compiler requires that the `main` method declare a `throws` clause. This time, however, it must declare `throws Throwable`, and in the absence of that, line n1 will again fail to compile.

You might think this last example is strange: After all, why doesn't the compiler see that the reality has not changed? The code clearly (to humans) won't throw any checked exceptions, right?

Well, rules are rules, and compiler optimizations are very hard to create, especially because the logic that must be applied to identify those optimizations' applicability increases. The reality is that if, and only if, you simply catch the exception and rethrow the same variable *with no possibility* of change (as evidenced by never making *any* assignment to the caught variable), then you get the benefit of that potential optimization. Unless all those conditions are met, the compiler decides the situation is too complex to analyze, and it doesn't apply that specific optimization.

So, now you know that the code compiles correctly, you can see what output it creates.

Option A describes the output that would result if `args` had a `null` value. However, when you pass zero arguments to a Java program, you get an array of zero length, not a `null` pointer. So the `NullPointerException` never arises, and option A is incorrect.

Because the length of the array is zero, the code will throw the `IllegalArgumentException`. This is caught by the matching `catch` block, `IAE` is printed, and the code rethrows the same exception. Next, the `finally` block is executed, which prints `F` and the unhandled (rethrown) exception propagates up to the caller of the `main` method.

That caller is the JVM itself, which responds by printing the stack trace before it exits to the operating system. This means option C is correct and option B is incorrect.

Conclusion: The correct answer is option C.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.